AD-A037 019 NEW JERSEY INST OF TECH NEWARK F/G 12/1

AN IMPROVED DIRECT SEARCH NUMERICAL OPTIMIZATION PROCEDURE.(U)

FEB 77 M PAPPAS N00014-75-C-0987

UNCLASSIFIED NJIT-NV-10 NL

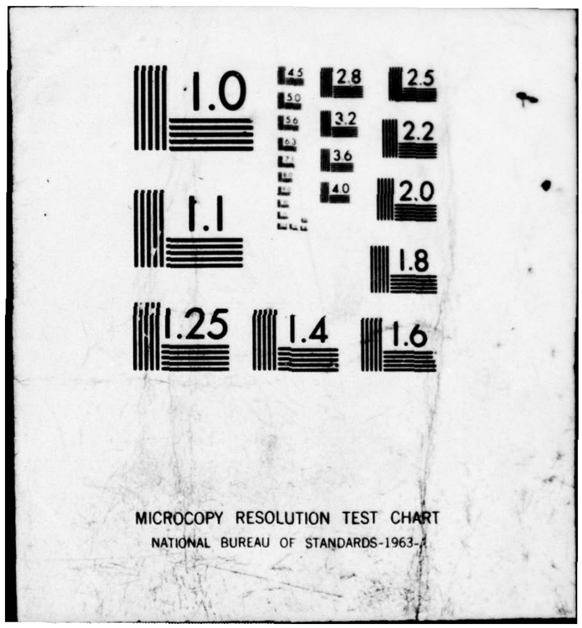
1 OF 1

AD A037019

END

DATE FILMED

4-77


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Contract No. ONR N00014-75-C0987-2

(12)

AN IMPROVED DIRECT SEARCH NUMERICAL OPTIMIZATION PROCEDURE

by

Michael Pappas

February 1977

NJIT Report No. NV 10

New Jersey Institute of Technology
323 High Street
Newark, New Jersey 07102

ABSTRACT

An improved, nonlinear, constrained mathematical programming optimization algorithm is presented in this report. It couples a rotating coordinate pattern search with a feasible direction finding procedure used at points of pattern search termination. The procedure is compared with nineteen algorithms, including most of the popular methods, on ten test problems. These problems are such that the majority of codes failed to solve more than half of them. The new method proved superior to all others in the overall generality and efficiency rating, being the only one solving all problems. It was particularly effective on constrained problems where it was best in all rating categories.

NOMENCLATURE

B_j	=	behavior of constraints $g_j(x_i)$
e_1, e_2, e_3	=	small arbitrary constants defining finite difference step size used to calculate the gradient of the objective function, nearness to constraints, and convergence criterion respectively.
$fa, \bar{f}a$	=	efficiency ratings (eqs. 30 and 31)
$f(x_i)$	=	objective function
$F(x_i)$	=	composite objective function
$g_j(x_i)$	=	constraint function
I	=	number of variables
J	=	number of constraints
K_A^-, K_A^+	=	set of active lower and upper regional, constraints, respectively
L_j	=	lower limit on the behavior of B_j
n_a, N_A	=	generality ratings (eq. 29)
$P(x_i)$	=	penalty function used when a constraint is violated
s_i	=	best feasible direction vector
T_A	=	generality and efficiency rating (eq. 32)
U_j	=	upper limit on the behavior of B_i
x_i	=	problem variables or variable vector
$\bar{x}_i$	=	optimal variable values
W_j	=	weighting factor for constraint $g_j(x_i)$ used in the feasible direction finding problem

α	=	step size
α_{min}	=	minimum step size
ε_j	=	constraint activity limits for the direction finding problem
σ	=	objective function and variable in the feasible direction finding problem
∇	=	gradient operator
$\lvert\phi\rvert$	=	magnitude of ϕ

Superscripts

ℓ	=	lower limit
r	=	at rth iteration
t	=	local exploration base
T	=	transpose of vector
u	=	upper limit
*	=	comparison quantity

INTRODUCTION

A variety of methods involving ordinary and variational calculus, mathematical programming, and a number of special techniques, such as the fully stressed concept used in structural design, are available to treat optimal design problems. Among these methods, the mathematical programming procedures appear to have the broadest range of application [1][1]. Such methods are flexible, easy to adapt, and can offer "automatic" optimal computer solutions [2].

Mathematical programming methods solve, or approach the solution to, the problem: Find those values of the variables, x_i, that minimize (or maximize) the objective function

$$f(x_i) \qquad i = 1,2, \ldots I \tag{1}$$

such that all constraints

$$g_j(x_i) \leq 0 \qquad j = 1,2, \ldots J \tag{2}$$

are satisfied. The objective function (often called the merit or payoff function) quantitatively defines the merit of the design as a function of the design variables, x_i. Inequality constraints are used to control or define the behavior of the design (behavior constraints).

Inequality constraints can also be used to limit the values of the variables within specified limits (regional constraints).

---

[1] Numbers in brackets designate References at end of report.

It is usually convenient to treat such constraints somewhat differently than behavior constraints. Regional constraints can be given in the form

$$x_i^{\ell} \leq x_i \leq x_i^u \qquad (3)$$

where x_i^{ℓ} and x_i^u are the lower and upper limits respectively on the variable x_i.

A number of effective methods exist for special forms of this problem, such as the unconstrained problem, the linear programming problem, and many quadratic programming problems. Relatively few effective methods are available, however, to treat the form most often encountered in design, the nonlinear constrained optimization problem. Even the better nonlinear prgramming methods have limitations on the size and complexity of problems they can solve within a justifiable amount of computer time. Large order, highly nonlinear systems requiring lengthy design computations can be quite formidable. A typical nonlinear mathematical programming procedure requires several hundred to several thousand sets of objective and constraint function evaluations to approach the optimum on most multivariable problems. Furthermore, reliability is a major problem with these methods [3].

The new Direct Search-Feasible Direction (DSFD) algorithm appears to offer superior performance with respect to speed and reliability [4]. It is the only procedure studied that solved all of Eason and Fenton's test problems [3-4].

No attempt was made in the original work on this algorithm, as reported in [4], to maximize its effectiveness. Rather the performance of the basic procedure was studied and presented.

It later became apparent that it was possible to substantially improve the speed of the procedure by eliminating needless computations and by other detailed improvements. This report presents and evaluates the latest version of the DSFD algorithm where such improvements have been incorporated.

THE OPTIMIZATION ALGORITHM

The basic search

The "rotating coordinate" (RC) pattern search used in [5] and described in detail in [6] is employed as the basic optimal search technique. The usual pattern search strategy [7] is utilized in this procedure except that after establishing a pattern move, the first local exploration step is taken in the direction of the move and all other local steps are taken normal to this direction rather than in the direction of the variable coordinates, as in the usual pattern search procedure. The movement of this search on a hypothetical composite objective surface using the penalty function of [5] is shown in Fig. 1. It may be seen from Path 1 that in the feasible region the search tends to move essentially in the direction of the gradient, while near the feasible-infeasible boundary it moves easily along the ridge. The local steps taken

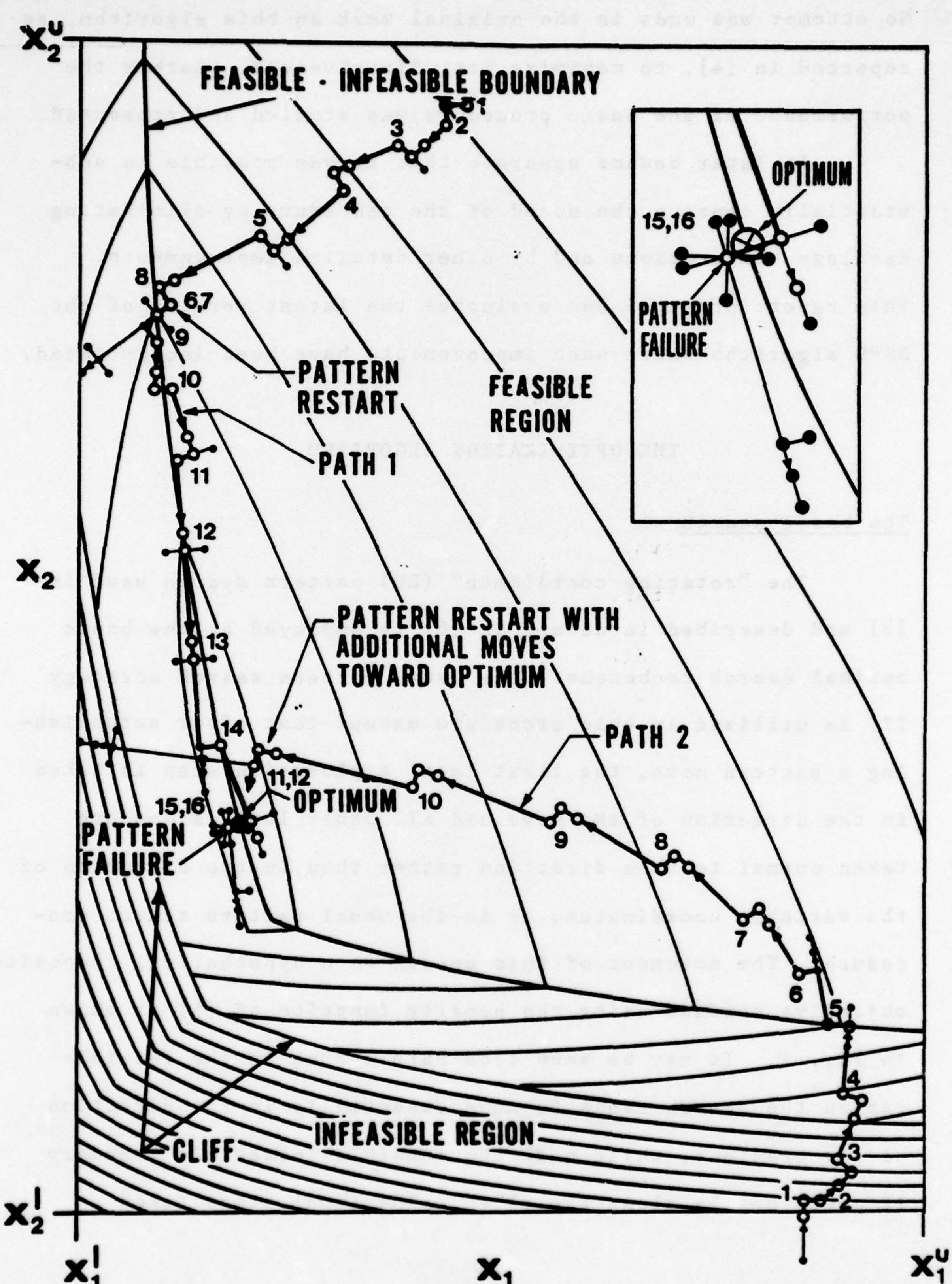


Figure 1. Movement of Rotating Coordinate Search

normal to the pattern move provide the turning component necessary to maintain efficient movement. It should be noted that the method does not require a feasible starting point as do most procedures (see Path 2). The closed dots in this figure indicate unsuccessful steps while the open dots are successful moves. The digits, associated with some open dots, identify the pattern base numbers.

This basic search procedure was found to be substantially superior to the original Hooke and Jeeves search in earlier comparison studies [5-7].

Since the basic search scheme is formulated to treat unconstrained optimization problems, the constrained problem of (1) - (3) is transformed to the form; Find

$$F(\bar{x}_i) = \min F(x_i) \tag{4}$$

where

$$F(x_i) = f(x_i) + P(x_i). \tag{5}$$

Here $P(x_i)$ is the largest of the penalties $p_j(x_i)$ which are of the form

$$p_j(x_i) = -\lambda_j[g_j(x_i)] \tag{6}$$

where

$$\begin{aligned} [\phi] &= \phi \qquad \phi < 0 \\ [\phi] &= 0 \qquad \phi > 0. \end{aligned} \tag{7}$$

When $0 > g_j > -e_2$, then

$$\lambda_j = 2[f(x_i) - f(x_i + \Delta x_i)]/[g_j(x_i) - g_j(x_i + \Delta x_i)] \tag{8}$$

where

$$\Delta x_i = e_1 \nabla f(x_i^t) / |\nabla f(x_i^t)|. \tag{9}$$

The quantity e_1 is an arbitrary small real number representing the size of the step, Δx_1, taken in the direction of the gradient of $f(x_i)$ where this gradient is evaluated at the local exploration base point x_i^t. It may be seen that a penalty is used only if a constraint is violated. If the violation of any of these constraints is greater than an arbitrarily specified amount e_2, then λ_j is not calculated from eqn (8) but is made arbitrarily large. That is, when

$$\left.\begin{aligned} g_j(x_i) &< -e_2 \\ \lambda_j &= K \end{aligned}\right\} \tag{10}$$

where K is an arbitrary large number.

The justification for the use of this penalty function form is discussed in detail in Ref. [5].

Violations of eqns (3) are handled by prohibiting moves that violate any of these equations.

All of the constraints are evaluated only at each temporary base (points reached after a pattern move). For all other points in the search only nearby constraints, this is those where

$$g_j(x_i) \geq e_2 \tag{11}$$

are evaluated since it is presumed that only these constraints need be considered.

The Secondary Search

Unfortunately, the pattern search, even with the rotating coordinate improvement, is not particularly reliable [5,6]. Thus, some method is needed to determine if the point of pattern search termination is optimal and if it is not, to determine a direction in which to restart the search. Pappas and Amba-Rao [8] and Siddal [9] apply a secondary search at points of pattern search termination. Unfortunately, the earlier secondary search strategies although more powerful locally than the pattern search do not provide a rigorous procedure for confirming optimality.

The direction finding procedure of Zoutendijk [10] can, however, be used to confirm the optimality of a point of pattern search termination and determine a direction in which to restart the pattern search if the point is not optimal. The direction finding problem can be stated: Given the set x_i, find the set s_i that results in a

$$\max \sigma \tag{12}$$

for which

$$\sigma > 0 \tag{13}$$

$$(s_i)^T \nabla f(x_i) + \sigma < 0 \tag{14}$$

$$(s_i)^T \nabla g_j(x_i) + W_j \sigma \leq 0 \quad j \in J_a \tag{15}$$

$$-1 \leq s_k \leq 0 \quad k \in K_a^- \tag{16}$$

$$0 \leq s_k \leq 1 \quad k \in K_a^+. \tag{17}$$

Here $(s_i)^T$ indicates the transpose of vector s_i, W_j is a weighing parameter, the set J_a contains the active constraints

$$g_j(x_1) > -\varepsilon_j \qquad (18)$$

where ε_j, an array of small arbitrary positive constants defining "activity," and K_a^- K_a^+ constitute the active upper and lower side constraints, respectively.

Zoutendijk [10] shows that if the solution s_i is a null vector, then the point is a local optimum and if not then the direction of s_i is the best feasible direction.

Equations (12)-(17) constitute a linear programming problem with the variable s_i and σ. Equation (12) is the objective function and the remaining equations the constraints. The solution s_i can be obtained reliably and efficiently using any suitable linear programming method such as the simplex procedure.

The Optimization Procedure.

The general procedure here is similar to that used in [3] and [6] except the pattern search and optimality checks used in [5] are replaced by those described previously. An arbitrary initial starting point x_i^0 is selected, F^* defined as equal to $F(x_i^0)$, and a quantity D^* set equal to unity. The optimal search is then started using the RC procedure with a step size α^0 and continued until it terminates at some point x_i^r, where the secondary search is invoked.

If the pattern search terminates at a point in the feasible region, that is, where equation (3) is satisfied for all j, the direction finding problem is formulated with $W_j = 1$ and solved at the point x_i^r. The components of the gradients are

generated using simple forward differences and thus no analytical derivatives are employed.

If the pattern search terminates in the infeasible region at a point near the feasible-infeasible boundary, that is, if some

$$e_2 \geq g_j(x_i^r) \geq 0 \tag{19}$$

(see Base 15 of Path 1 in Fig. 1) a weighting parameter of $W_j = 100$ is used to help drive the search to the feasible region.

In the case where the pattern search terminates in the infeasible region away from the boundary, that is, where any constraint equation violates the left side of (19), the search is abandoned since in this instance the pattern search has failed to generate a feasible point, even with the use of a large penalty parameter.

Once the direction vector s_i is computed, the quantity $F(x_i^{r+1})$ is evaluated, where

$$x_i^{r+1} = x_i^r + \Delta x_i \tag{20}$$

and

$$\Delta x_i = \alpha^r s_i / \left| \left[\sum_{m=1}^{I} s_m^2 \right]^{1/2} \right| \quad \text{if } s_i \neq 0 \tag{21}$$

Here α^r is the step size used for the pattern search just terminated. If any constraint not considered in the formulation of the direction finding problem has been violated at x_i^{r+1}, then the direction finding problem is reformulated considering these constraints and a new s_i and $F(x_i^{r+1})$ computed.

If $s_i = 0$ the activity definition limits ϵ_j and the step size α^r are halved. If the set J_a changes as a result of the

change in ε_j due to one of the constraints becoming deactivated as a result of the reduction in the activity specification the direction finding problem is reformulated and solved. Otherwise the activity limit and step size is further reduced. This process is repeated until a nonzero solution of s_i is obtained or until

$$\alpha^r \le \alpha_{min} \tag{22}$$

where α_{min} is a specified minimum basic step size.

Now if

$$F(x_n^{r+1}) < F(x_n^r) \tag{23}$$

the RC search is restarted using x_n^{r+1} as the new base point in the pattern search strategy. If equation (22) is not satisfied, then α is refined as

$$\alpha^{r+1} = \alpha^r/2 \tag{24}$$

Now if

$$\alpha^{r+1} < \alpha_{min} \tag{25}$$

or if

$$\left.\begin{aligned} & D^r \le e_3 \\ \text{and} \quad & D^r < D^* \\ \text{where} \quad & D^r = [F^* - F(x_i^r)]/F^* \end{aligned}\right\} \tag{26}$$

and where α_{min} and e_3 are arbitrary small positive variable and objective function convergence constants, respectively, the search is abandoned and the point is assumed to be sufficiently near optimum. Otherwise a new x_i^{r+1} is defined by equations (20-21) with α^r replaced by α^{r+1} until equation (23), (25), or (26) is satisfied.

Once equation (23) is satisfied, then the RC search is restarted, with x_i^{r+1} as a new base point in the pattern search strategy, with step size α^{r+1}, $F^* = F(x_i^r)$, and $D^* = D^r$. If, however, two successive step size reductions fail to satisfy equation (23), then the entire optimization procedure is restarting from point x_i^r, with step size α^{r+1}, $F^* = F(x_i^r)$, and $D^* = D^r$ and repeated until equation (25) or (26) is satisfied.

It may be seen that the step defined by equation (21), taken when s_i is nonzero, is a move in the best feasible direction. Thus, if equation (23) is not satisfied by this move, it indicates that the step size is too large. The step size is then reduced in an effort to locate a better point, first by taking a smaller step in the s_i direction, and in the event this fails, by attempting to restart the pattern using this smaller step. The process is repeated until convergence of the objective function is achieved or a minimum step size is reached.

A zero s vector indicates the presence of an optimum. Since, however, in the consideration of the active constraints, $\varepsilon_j \neq 0$, the point x_i^r may be merely near, rather than at, the optimum. Thus, the activity limit is reduced when a null s vector solution is encountered and the Feasible Direction problem reexamined to determine if the point is indeed an optimum. Since in this procedure

$$\varepsilon_j = C_1 \, \alpha^r/\alpha^0 \qquad (27)$$

when equation (22) is satisfied it is assumed that the ε_j are sufficiently small to allow the point x_i^r to be considered an optimum.

This procedure is the same as that of reference [4] except for the following differences. In reference [4];

1. the values of $\nabla f(x_i)$ were evaluated at each point calling for the computation of a penalty parameter λ_j. Thus $\nabla f(x_i)$ is often calculated several times during a local exploration. Since, however, only an estimate of ∇f is required here, $\nabla f(x_i^r)$ is computed here only once during local explorations. A substantial reduction in the number of required objective function evaluations required for penalty evaluations is thereby achieved.
2. all constraints were evaluated at all points. Since, however, only those constraints which are active or near active influence movement toward the optimum at a given point, the procedure used here only considers such constraints. All constraints are evaluated only periodically (after each pattern jump) to monitor constraint activity.
3. at points of pattern search termination in the unfeasible region near the feasible-infeasible boundary the pattern search was restarted using $\lambda_j = K$, the large penalty parameter, in an attempt to drive the search to the feasible region. In the strategy used here, movement in a direction finding problem with $W_j = 100$, is utilized.
4. slightly different version of the Rotating Coordinate Pattern Search is used. The version employed here contains several improvements.

COMPARISON STUDY

The work of Eason and Fenton [3] forms the primary basis for the comparison between the algorithm presented here and other optimization procedures. A FORTRAN IV code called CADOP3 based on this algorithm was developed and used on the test problems given in [3]. The code is operational with IBM FORTRAN G and H and the UNIVAC TDOS and TSOS systems. This code required the user to supply only the objective and constraint functions, the initial values of the variables, and regional constraint values, if used. The convergence criteria and initial step size may also be specified.

If no step size is specified, the size is internally generated. In this program the initial step size is selected such that at the starting point a step of size α^0 in the ∇f direction produces a one percent change in the value of the objective function. The internally generated step size is constrained so that $\alpha^0 > 0.005$. The minimum step size is defined as $\alpha_{min} = \alpha^0/1000$ unless specified otherwise by the user.

Nondimensional constraint equations of the form

$$\left.\begin{array}{l} g_j(x_i) = (B_j - U_j)/|U_j| \leq 0 \text{ when } U_j \neq 0 \text{ and } B_j \neq 0 \\ \text{or} \\ g_j(x_i) = (L_j - B_j)/|L_j| \leq 0 \text{ when } L_j \neq 0 \text{ and } B_j \neq 0 \end{array}\right\} \quad (28)$$

are used. Otherwise a dimensional form is used.

Here B_j represents the constraint "behavior" and U_j and L_j the upper and lower limits on this behavior, respectively. The user supplies the expressions for $f(x_i)$ and the expressions or parameters defining B_j, U_j, and L_j.

A constraint given as $0 \leq b(x_i) \leq A$ would be written as two constraint equations. For example, one could be of the form of the first of equations (28) where $B_1 = b(x_n)$ and $U_1 = A$. The second would be of the form $g_2 = - b(x_n)$ since $L_1 = 0$.

The program uses $e_1 = 0.0001$, $e_2 = 0.10$ and $e_3 = 1 \times 10^{-7}$. Multiple starting points can be tried in a single computer run to provide additional optimality confirmation. Such a technique is recommended even though the algorithm presented here seems reasonably reliable, since no nonlinear method can be considered absolutely reliable.

All ten problems of reference [3] were run with the CADOP3 code from the starting points specified in reference [3]. The internally generated step sizes were used for all problems and no algorithm control constants were changed during the study. Thus, there was no tuning of the code to individual problems. All problems were run on an IBM 370 model ~~91~~ 168 using the "G" level compiler and double precision. Thus, the runs closely simulate those of the Eason and Fenton study which likewise used an IBM machine with the same level compiler and precision specification. Thus the comparisons can be considered accurate.

Table 1 briefly describes the codes studied in reference [3] with the addition of the DSDA code, the earlier CADOP2 code, and the CADOP3 code studied here. Additional details on these codes can be found in references [4] and [11]. Table 2 presents results of the performance of these codes on the ten test problems in the form of a success matrix. The numerical entries indicate the normalized time required for solution [3], the symbol

TABLE 1

CODE DESCRIPTIONS

Name	Description	Algor class*
ADRANS	Random search followed by pattern moves	DS
CLIMB	Rosenbrock search	DS
DAVID	Davidon-Fletcher-Powell with numerical derivatives	GF
DFMCG	Fletcher-Reeves conjugate gradient method with secant approximation derivatives	G
DFMFP	Davidon-Fletcher-Powell with secant approximation derivatives	G
FMIND	Hook & Jeeves pattern search	DS
GRADA4	Steepest descent method	G
GRID1	Grid and star network search, with shrinkage	AR
MEMGRD	Davidon-Fletcher-Powell with retained step length information	GF
NMSERS	Simplex search	DS
PATSH	Modified pattern search	DS
PATRNO	Modified pattern search, dome strategy	DS
PATRN1	Modified pattern search, ridge strategy	DS
RANDOM	Random search with shrinkage	AR
SEEK1	Pattern search followed by random search	DS
SEEK3	Modified pattern search	DSF
SIMPLX	Modified simplex search	DSF
DSDA	Modified pattern search followed by Mugele's search	DS
CADOP2	Direct search-feasible direction algorithm	DS
CADOP3	Refined direct search-feasible direction algorithm	DS

*DS = direct search, DSF = direct search employing SUMT strategy and penalty function, G = gradient procedure, GF = gradient procedure using SUMT strategy and penalty function. AR = area reduction method.

TABLE 2

PERFORMANCE OF OPTIMIZATION CODES*

Problem No.	1	2	3	4	5	6	7	8	9	10
Variables	5	3	5	4	2	2	3	2	2	4
Constraints**	10	2	6	0	0	1	2	0	6	0
Code names										
ADRANS	2.64	0.458	1.65	P	0.100	0.654	0.159	0.069	P	P
CLIMB				0.015	0.005			0.007		
DAVID		0.188	0.84	1.132	0.075	0.046				
DFMCG			P	0.015	P			0.004		
DFMFP			P	0.038	0.250		P	0.387		
FMIND		0.004	P		0.140		0.003	0.003	P	3.74
GRAD4		P	P	0.283	0.393		P	0.004		P
GRID1	P	P		0.033	P	P		0.037	P	P
MEMGRD		0.143	0.91	0.059	0.066	0.067	P			
NMSERS		0.019	0.045	0.060	0.002	0.012	P	0.007	0.39	P
PATSH	1.78	0.220	1.00	0.013	0.018	0.020	0.010	0.009	P	3.54
PATRNO	P	P	P	0.021	P			0.002	P	1.95
PATRN1	P	P	P	0.008	0.002	P		0.001	1.02	1.20
RANDOM		P		P		0.024	1.20	0.013		P
SEEK1	P	P	P		0.010		0.010	0.007	P	1.53
SEEK3		0.102	0.14	0.212	0.035	0.191	0.141	0.013	4.20	P
SIMPLX	2.97	0.297	1.31	0.196	0.035	0.191	0.262	0.050	P	
DSDA	P	0.021	0.047	0.104	0.003	0.008	0.145	0.004	1.94	1.35
CADOP2	0.307	0.016	0.090	0.075	0.008	0.011	0.069	0.003	1.55	1.73
CADOP3	0.119	0.006	0.030	0.080	0.004	0.011	0.025	0.005	1.00	1.60

*Numerical entry indicates normalized time required for solution, and P indicates progress toward a solution [3, 12].

**Excluding regional constraints.

"P" indicates progress toward a solution, and a blank indicates failure. Definitions of "solution" and "progress" vary for each problem and are given in reference [11] as well as a detailed description of each problem.

The data from Table 2 may be applied to a number of rating schemes for comparing the codes tested. Table 3, presents relative rankings of the codes using some of the rating criteria of reference [3]. The rating equations used are as follows:

The number of problems solved by code "a" is n_a and

$$N_a = n_a + n_a'/2 \tag{29}$$

where n_a' is the number of problems in which a P rating was achieved. The efficiency ratings are given by

$$f_a = \left[\sum_{p=1}^{10} b_{ap} t_{ap} / \min(t_{ap}) \right] / n_a \tag{30}$$

where $b_{ap} = 1$ if code "a" solved problem "p" and zero otherwise, t_{ap} is the normalized time required for solution (cpu time divided by cpu time required to run a timing standardization program) [3], and $\min(t_{ap})$ is the shortest time required by any of the codes to solve problem "p." The other efficiency criterion is

$$\overline{f}_a = \left[\sum_{p=1}^{10} b_{ap} t_{ap} / \mathrm{mean}(t_{ap}) \right] / n_a \tag{31}$$

where mean (t_{ap}) is the average time required by the codes to solve problem "p." Here the overall rating number for generality (reliability) and efficiency (speed) is given by

$$T_a = \sum_{p=1}^{10} t_{ap} \tag{32}$$

TABLE 3

RELATIVE RANKING OF OPTIMIZATION CODES

	Generality			Efficiency				Generality and Efficiency	
	n_a		N_a	f_a		$\bar{f}_a$		T_a	
10	CADOP3	10	CADOP3	1.3	PATRN1	0.17	NMSERS	3	CADOP3
	CADOP2		CADOP2					4	CADOP2
				3.4	CADOP3	.25	CADOP3		
9	DSDA	9.5	DSDA	3.5	NMSERS	0.26	PATRN1	9	DSDA
	PATSH		PATSH	5.6	CADOP2				
				8.9	DSDA	0.34	CADOP2	15	PATSCH
8	SEEK3		SEEK3						
	SIMPLX	8.5	SIMPLX			0.41	DSDA	16	NMSERS
			ADRANS						PATRN1
7	ADRANS						FMIND		
	NMSERS	8.0	NMSERS	15	PATSCH	0.77	PATSCH		
								18	SEEK3
	PATRNI	7.0	PATRN1	16	FMIND	0.90	NEMGRD		
5	FMIND		FMIND	20	SEEK3			21	SIMPLX
	MEMGRD	6.0	SEEK1	22	NEMGRD				
	DAVID							24	ADRAMS
						1.0	SEEK3		
4	SEEK1		NEMGRD						
		5.5	PATRNO	42	SIMPLX			25	FMIND
	CLIMB					1.5	SIMPLX		
	DFMFP		DAVID	51	DAVID			26	NEMGRD
3	GRAD4		GRAD4			2.4	DAVID		
	PATRNO	5.0	GRID1	60	ADRANS			27	DAVID
	RANDOM		RANDOM			2.5	ADRANS		
2	DFMCG	4.0	DFMFP						
	GRID1								
		3.0	CLIMB						
			DFMCG						

where t_{ap} is set equal to twice the time used by the slowest code solving a problem "p" that code "a" could not solve. This penalty time is used to penalize code unreliability. Only codes that solved half or more of the problems are rated for efficiency.

The tables presented here are essentially similar to those given in reference [3], except that the DSDA, CADOP2 and CADOP3 codes are included. Thus, both the rating schemes and presentation of results are essentially those of [3].

Based on its performance on the ten problems of reference [3] the new algorithm appears to be a relatively fast and reliable optimization procedure. Codes based on this method were the only ones solving all problems attempted. CADOP3 is comparable in speed to the faster codes (NMSERS, and PATRN1). It was significantly faster than average in all problems and was the fastest code solving problems 1 and 2. Compared to the relatively reliable codes ($n_a \geq 8$), CADOP3 was faster than DSDA on seven of nine problems solved by DSDA, faster than CADOP2 on seven of ten, faster than PATSH on seven of nine, and faster than SEEK3 and SIMPLX in all problems solved by these schemes. As may be seen that DSDA and CADOP2 are the only reasonably reliable codes comparable in speed to CADOP2. CADOP3 appears significantly faster than PATSCH, SEEK3, and SIMPLX. Only the straight (PATRN1) and simplex (NMSERS) codes were generally faster. Of this group, only NMSERS appears to be sufficiently reliable to merit serious consideration. The differences in speed between

NMSERS and CADOP3 is, however, overshadowed by the superior reliability of the latter. Thus, in the overall generality and efficiency rating, CADOP3 stands alone. Viewed on the basis of these comparisons, CADOP3 appears to be a superior nonlinear MP code.

It should be noted that problems 4, 5, 8 and 10 are unconstrained at the optimum points. Thus, even though problem 10 has a regional constraint which is active at the start of the search, these problems can be considered essentially unconstrained.

Those codes (PATRN1 and (NMSER) which are faster overall than CADOP3 obtain their superior efficiency ratings primarily from faster solutions of such problems. The DSFD algorithm is designed, however, for constrained problems. Thus, although its overall performance on constrained and unconstrained problems, or its performance on unconstrained problems, is of interest, a direct evaluation of its performance on constrained problems is also appropriate. Thus a comparison on those problems with behavior constraints (problems 1, 2, 3, 6, 7, and 9) is given in Table 4.

In Table 4 only those codes that solved three or more of the constrained problems are evaluated for efficiency. In this comparison CADOP3 is comparable to NMSERS in speed with these codes being substantially more efficient than any other code. CADOP3 is ranked much higher in the overall rating since NMSERS failed on two of the six problems. CADOP3 is faster than CADOP2 and DSDA on all but one problem. This problem has only one constraint, however, and thus the reduction associated with checking only nearby rather than all constraints is lost.

TABLE 4

RELATIVE RANKING OF OPTIMIZATION CODES ON PROBLEMS

WITH BEHAVIOR CONSTRAINTS

Generalty				Efficiency				Generalty and Efficiency	
n_a		N_a		f_a		$\bar{f}_a$		T_a	
6	CADOP3	6	CADOP3	2.2	NMSERS	0.14	NMSERS	1.2	CADOP3
	CADOP2		CADOP2	2.6	CADOP3	0.16	CADOP3	2.0	CADOP2
5	DSDA	5	DSDA					8.1	DSDA
	PATSH		PATSH	6.3	CADOP2	0.30	CADOP2		
	ADRANS		ADRANS	12.	DSDA	0.44	DSDA	8.8	NMSERS
	SEEK3	5	SEEKS	22	PATSH	0.93	PATSH	11	PATSH
					SEEK3	1.1	NEMGRD		SEEK3
4	NMSERS	4.5	NMSERS				DAVID	13	SIMPLX
		3.5	SIMPLX	25	NEMGRD				NEMGRD
3	MEMGRD		MEMGRD	27	DAVID	1.2	SEEK3	14	ADRAMS
	SIMPLX					1.9	SIMPLX		
	DAVID		DAVID	51	SIMPLX			18	DAVID
		3.0	FMIND	65	ADRANS	2.9	ADRANS		
			SEEK1						
2	FMIND		PATRN1						
	RANDOM	2.5	RANDOM						
1	PATRN1	2.0	PATRNO						
	SEEK1		GRID1						
		1.5	GRAD4						
0	PATRNO	1.0	DFMFD						
	GRIDI	0.5	DFMCG						
	GRAD4	0.0	CLIMB						
	DFMCG								
	DFMFD								
	CLIMB								

CADOP3 is faster than NMSERS on three of four problems although both codes are essentially similar in speed. It is faster on all problems than the remainder of the codes rated for efficiency. CADOP3 is first in the generality and efficiency rating since it is faster than CADOP2 and since all other codes were penalized in this rating for failure on one or more problems. It may be seen therefore that the DSFD improvements cited here did indeed increase performance.

Thus the improved DSFD procedure seems, on the basis of this comparison, clearly superior to the other procedures studied on constrained problems.

Table 5 lists the number of function evaluations required by CADOP3 to reach solution. The comparison data of reference [3] does not provide these values for the other codes. These values are provided here since it is felt that for most engineering problems the number of function evaluations is a better criterion of performance than nondimensional time.

The time required for solution can be viewed as the sum of the function evaluation and algorithm execution time. For very simple functions most of the required time may be associated with algorithm execution particularly where the algorithms are complex. For such problems, however, time is usually of little importance since any reasonably effective procedure can generate a relatively low cost solution. On the other hand, efficiency becomes important where function evaluation is computationally demanding. Here algorithm execution time may be quite small

TABLE 5

NUMBER OF FUNCTION EVALUATIONS FOR SOLUTION

Problem No.	Objective Function Evaluations	Constraint Function Evaluations
1	2088	4860
2	266	190
3	622	993
4	4910	--
5	347	--
6	378	322
7	556	737
8	217	--
9	488	447
10	849	--

compared to function evaluation time. In such cases the number of function evaluations is a reasonably accurate criterion for performance.

Preliminary experiments indicate that although nondimensional time is a valid comparison criterion where codes are run using the similar compilation software it is not valid otherwise. It is felt therefore that the number of function evaluations is a superior comparison criterion since it is a reliable indicator of performance on problems where efficiency is important. Thus this information is provided here so as to allow future comparisons.

The principal apparent disadvantage of CADOP3 is its relative complexity compared to some of the other reasonably reliable methods. It contains 713 FORTRAN statements, while, for example, DSDA contains 372 and PATSH only about 75. Thus for simple problems of relatively low dimensionality, one of the simpler codes may be preferable.

It may be seen that the gradient-based procedures, including those employing the SUMT strategy and penalty function [12], so widely used in engineering problems, performed rather poorly. The best gradient-based code DAVID solved only half the problems. The direct search procedures in general provided much better performance although only four, CADOP3, CADOP2, DSDA, and PATSH, solved or approached the solution to all ten problems with only CADOP3 and CADOP2 solving all problems.

It should be noted, however, that this comparison employed rather small test problems (two to five variables with zero to ten behavior constraints). It is not clear that the direct search procedures would also possess reliability and speed superior to the gradient methods on large problems. Still, the superiority in this study of the better direct search procedures is dramatic while the performance of the gradient schemes is rather dismal. Furthermore, although the better direct search procedures may not be as reliable on larger problems, one would certainly not expect the reliability of the gradient procedures to improve on such problems.

CONCLUSION

Although it is recognized that such favorable findings on the CADOP3 code presented by its developer can be viewed with some skepticism, it should be noted that the performance cited is based primarily on rating schemes and problems selected by Eason and Fenton and not by the author. Furthermore, this performance was achieved with a conscious effort to avoid any tuning of the code to individual problems. Thus, these results indicate that the DSFD algorithm appears to be a superior nonlinear mathematical programming procedure at least on relatively small problems on which is was tested.

It should be noted, however, that this algorithm combines the strength of the direct search procedures (speed) and the Zoutendijk method (ability to confirm optimality and identify a feasible direction) in such a way that the weakness of the direct search in treating problems of large dimensionality is minimized. Thus, one would expect this new procedure to also be quite effective on larger problems. This potential has, however, yet to be demonstrated.

REFERENCES

1. Sheu, C. Y., and Prager, W., "Recent Developments in Optimal Structural Design," Applied Mechanics Review, Vol. 21, No. 10, Oct. 1968, pp. 985-992.

2. Schmit, L. A., "Automated Design," International Science and Technology, No. 54, June 1966, pp. 63-78 and 115-117.

3. Eason, E. D., and Fenton, R. G., "A Comparison of Numerical Optimization Methods for Engineering Design," Journal of Engineering for Industry, Trans. ASME, Series B, Vol. 96, No. 1, Feb., 1974, pp. 196-201.

4. Pappas, M. and Moradi, J. Y., "An Improved Direct Search Mathematical Programming Algorithm," Journal of Engineering for Industry, Trans. ASME, Series B, Vol. 97, No. 4, Nov., 1975, pp. 1305-1310.

5. Pappas, M., "Use of Direct Search in Automated Optimal Design," Journal of Engineering for Industry, Trans. ASME, Series B, Vol. 94, No. 2, May, 1972, pp. 395-401.

6. Pappas, M., "A Gradient-Direct Search Procedure for Automated Optimum Structural Design," Ph.D. dissertation, Department of Aerospace and Mechanical Engineering, Rutgers-The State University of New Jersey, Jan., 1970.

7. Hooke, R., and Jeeves, T. A., "Direct Search Solution of Numerical and Statistical Problems," Journal of Computing Machinery, Vol. 8, No. 2, Apr. 1961, pp. 212-229.

8. Pappas, M., and Amba-Rao, C. L., "A Direct Search Algorithm for Automated Optimal Structural Design," AIAA Journal, Vol. 9, No. 3, Mar. 1971, pp. 387-393.

9. Siddall, J. N., "OPTISEP" Designer Optimization Subroutines, ME/71/DSN/REP 1, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada, 1971.

10. Zoutendijk, G., Methods of Feasible Directions, Elsevier, Amsterdam, 1960.

11. Pappas, M., "Performance of the 'Direct Search Design Algorithm' as a Numerical Optimization Method," AIAA Journal, Vol. 13, No. 6, June 1975, pp. 827-829.

12. Eason, E. D., and Fenton, R. G., Testing and Evaluation of Numerical Methods for Design Optimization, UTME-TP 7204, University of Toronto, Canada, 1972.

13. Fiacco, A. V., and McCormick, G. P. Nonlinear Programming; Sequential Unconstrained Minimization Techniques, Wiley, New York, 1968.

APPENDIX A AND B

USER INSTRUCTIONS

and

PROGRAM LISTING

APPENDIX A

USER INSTRUCTIONS FOR CADOP3

1. Instruction

This code treates inequality constrained or unconstrained linear or nonlinear minimization problems by means of a direct search mathematical programming method. The method starts from a user specified starting or initial point and generates a sequence of better points until, hopefully, an optimal, or near optimal, point is reached. The code is intended primarily for constrained nonlinear problems. Linear problems are much more effectively handled by one of many linear programming methods. Unconstrained problems are better treated by UNCDP, a simplified version of CADOP3, or one of several other unconstrained nonlinear problem codes. CADOP3 will, however, treat both linear and unconstrained problems.

It should be noted that nonlinear mathematical programming methods generally do not guarantee an optimal solution because of the possibility of multiple local optima, algorithm failure, or numerical difficulties. Alghough CADOP3 is among the most reliable of the nonlinear mathematical programming codes it cannot be considered absolutely reliable. Thus, it is desirable to use several optimization runs with widely separated starting points to confirm the convergence has been achieved or to determine if local optima are present.

2. Problem Formulation

This code treats the problem:

Find those values $\bar{x}_i$ of the set of "variables" x_i, i = 1,2----IP and the constant "parameters" p_k, k = 1,2----KP, that result in the minimum value of the objective function f (p_k, x_i), that is

$$f\ (p_k, \bar{x}_i) = \min\ [f_k(p, x_i)] \tag{1}$$

while also satisfying the "behavior" constraints

$$g_j\ (p_k x_i) \leq 0 \qquad j = 1,2,\text{----}JP \tag{2}$$

and the "regional" constraints

$$x_i^{\ell} \leq x_i \leq x_i^{u} \tag{3}$$

Two types of behavior constraints are recognized, one type are "linear" constraints of the form

$$g_j\ (x_i) = \sum_{i=1}^{IP} a_{ij} x_i \tag{4}$$

Where f (x_i) is also of this form one has a linear programming problem. If any of Eqs. (1 and 2) is not of this form the problem is nonlinear. Constraints not of the form of Eq. (4) are nonlinear constraints.

The program in its present form treats problems with

$$2 \leq IP \leq 10, \quad 0 \leq JP \leq 10 \text{ and } \leq KP \leq 100$$

3. Specification of Objective and Constraint Functions

The subprogram FUNCTION OBJ(J) is essentially a dummy subprogram created to accept the users FORTRAN IV program statements defining the objective and behavior constraint functions. The normal form of the behavior constraints is

$$B_j (p_k, x_i) \leq U_j (p_k, x_i) \qquad (4)$$

where B_j can be thought of as the behavior and U_j the upper limit of behavior.

The FUNCTION OBJ(J) is modified so as to define the objective and constraint functions by inserting the following statements immediately after the COMMON statement

GO TO (1,2,----jp), J (omit if no constraints are used)

OBJ = expression defining f (p_k, x_i) using arrays P(k), X(i)

RETURN

j B = expression defining B_j

U = expression defining U_j

GO TO 101

A constraint statement group is used for every constraint. The last constraint group need not use the final GO TO statement.

Example

For the problem:

minimize

$$2x_1 x_2 - P_1 x_3 x_4^2$$

such that

$$2x_3^3 - x_1 \leq p_2$$

and

$$-6 \leq x_2^2 - x_4 \leq 0$$

one could define three constraints and insert the statements

```
GO TO (1,2,3),J
OBJ = 2. * X (1) * X (2) - P (1) * X (3) * X (4) **2
RETURN
1 B = 2.* X (3) ** 3 - X (1)
U = P(2)
GO TO 101
2 B = X(2) ** 2 - X (4)
U = 0.
GO TO 101
3 U = X (2) ** 2 - X (4)
B = - 6.
```

Alternately, noting that both upper and lower limits on the second constraint equation cannot be active simultaneously, one could use two constraint equations and insert the program segment:

```
      GO TO (1,2),J
      OBJ = 2. * X (1) * (2) - P (1) * X (3) * X (4) ** 2
      RETURN
    1 B = 2. * X (3) ** 3 - X (1)
      U = P
      GO TO 101
    2 TB = X (2) ** 2 - X (4)
      IF(TB.LT.-3.)  GO TO 3
      B = TB
      U = 0.
      GO TO 101
    3 B = -6.
      U = TB
```

The constraints values at the optimum are printed in the form of equation (2) where

$$g_j = \begin{cases} (B_j - U_j)/|U_j| & U_j \neq 0 \text{ and } B_j \neq 0 \\ (Bj - U_j) & U_j = 0 \text{ or } B_j = 0 \end{cases}$$

thus a negative value of g_j indicates the constraint is satisfied.

4. Data Cards

The program allows multiple optimization runs of a particular problem within a single computer run using different parameters, for parametric studies, different starting points, for optimality confirmation or search for local optima, and different sets of regional constraint limits.

The user must, by use of the data card set, specify the number of parameters, variables and behavior constraints used, identify the linear constraints, specify for the first run if regional constraints are to be used, and for subsequent runs whether the parameters, initial point or regional limits are to be changed. If parameters are used they must be entered. The initial point must be entered. The values for the regional constraints must be entered if such limits are imposed.

The first data set is entered on the "Problem Constants" card which specifies, in order, the number of parameters (KP), variables (IP) and behavior constraints (JP), used (see line 9 of program listing). The data is entered in 3I10 format in the first 3 fields in order and must be right justified.

If constraints are used (JP > 0) a second data set is entered on the "Constraint Identification" card which identifies the linear behavior constraints. If a constraint is linear a digit is entered in the column with the same number as the constraint. If no linear behavior constraints are used a blank card is employed (see line 10).

The third data set is entered on the "Data Control" card and is used to specify whether: 1) new parameters are to be used (ICNTR2), 2) a new initial point is to be used (ICNTR3) and, 3) regional limits or new regional limits are to be used (ICNTR4) (see lines 26-43). The entries are made in 3I10 format in the order given above. For the first run no entries in the first two fields are needed. If regional limits are used an entry, any nonzero digit, is made in the third field (col 21-30).

If and only if the number of parameters specified is greater than zero (KP > 0) a fourth data set is used to enter the problem parameters p_k. The entries are made 5 to a card in F15.5 format in order i = 1,2----KP, (see lines 21 and 22).

The fifth data set specifies the starting point and entries are made, in order, 8 to a card in F10.5 format (see lines 23 and 29).

If and only if an entry is made in the third field of the Data Control card (ICNTR4 = 0) a sixth data set defining the lower limits is entered, 8 to a card, in F10.5 format followed by an upper limit set (starting with a new card) with similar format. Both complete limit sets must be entered (see lines 30-34).

For every additional run an additional Data Control card is added followed by parmeter and/or initial point and/or regional limit sets, where and only where the need for a new set is indicated by an appropriate entry (ICNTR2 = 1 or (ICNTR3 = 1 or ICNTRL4 = 1) on the Data Control card (see lines 26-34).

A blank Data Control card is used to terminate the run(s) (see lines 26 and 27).

5. Conversion To Single Precision

The program as supplied is a double precision form. Experience has shown, however, that the great majority of problems can be treated adequately in single precision and thus it is recommended that single precision be tried first. To convert to single precision remove the IMPLICIT statement at the beginning of MAIN and all subprograms and the function conversion statements such as SQRT(B) = DSQRT(B) etc., near the beginning of most subprograms. Furthermore, the REAL FUNCTION OBJ*8(J) should be put in single precision form.

6. Change of Problem Size

To change the maximum number of variables (IP) or the maximum number of constraints (JP) the program can treat, change the arrays in MAIN and all subprograms as follows:

1. change all D, DZ, X, DL, and DU arrays in common to D(IP), DZ(IP) etc.,

2. change the G array (T in OPT2 and LINPRO) as well as the CK, IA, IC array in COMMON and the BL array in COMMON/OPT/ where they occur to G(JP), CK(JP) ETC.,

3. change the V array (A array in OPT2 and LINPRO) to V(M,N) in COMMON where

$$M = JP + 4IP + 5$$

$$N = JP + 5IP + 6$$

4. change all DUMMY arrays such as DUM, IDUM, etc., to equalize common sizes.

5. In the OPT2 SUBROUTINE DIMENSION statement change DT, SS, IB, IC, to DT(IP) etc., change GG to GG(JP), TN to TN(JP+1), ID to ID(JP), DK to DK(JP+1, IP), DBJ to DBJ (JP+1, IP) and SSS to SSS(M) where M is as above.

6. In the FIND SUBROUTINE DIMENSION statement change DT(10) to DT(IP).

7. In the PATTRN SUBROUTINE DIMENSION statement change all 10's to IP.

APPENDIX B

CADOP3

PROGRAM LISTING

```
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON D(10),P(100),DZ(10),X(10),G(10),A,AMIN,DL(10),DU(10),
     1V(55,66),CK(10),Q,IP,JP,IA(10),KP,KW,KL,IK,IDUM(21),JX
      COMMON/OPT/BL(10),TN
1     FORMAT(5F15.5)
2     FORMAT(4I10)
3     FORMAT(2F10.5)
101   FORMAT(8F10.5)
      READ2,KP,IP,JP
      IF(JP.NE.0)READ 44,(CK(K),K=1,JP)
      IF(JP.EQ.0)GO TO 45
      DO 46 K=1,JP
      IF(CK(K).NE.0.)CK(K)=0.01
      IF(CK(K).EQ.0.)CK(K)=1.
      IF(CK(K).LT..5)CK(K)=0.
   46 CONTINUE
   45 DO 21 I=1,IP
      DL(I)=-1.E+48
44    FORMAT(80F1.0)
21    DU(I)=1.E+48
      READ2,ICNTR2,ICNTR3,ICNTR4
      IF(KP.EQ.0) GO TO 12
      READ1,(P(I),I=1,KP)
12    READ101,(D(I),I=1,IP)
18    FORMAT(1H0,' STARTING DESIGN VARIABLE VALUES' )
      GO TO 23
24    READ(5,2,END=33)ICNTR2,ICNTR3,ICNTR4
       IF(ICNTR2+ICNTR3+ICNTR4.EQ.0)STOP
       IF(ICNTR2.NE.0)READ1,(P(I),I=1,KP)
      IF(ICNTR3.NE.0)READ101,(D(I),I=1,IP)
23    IF(ICNTR4.EQ.0)GO TO 22
      READ 101,(DL(I),I=1,IP)
17    FORMAT(1H0,' LOWER LIMITS OF DESIGN VARIABLES')
       READ 101,(DU(I),I=1,IP)
19    FORMAT(1H0,' UPPER LIMITS OF DESIGN VARIABLES')
22    IF(KP.LE.0) GO TO 104
      PRINT 6
      PRINT7,(K,P(K),K=1,KP)
104    PRINT 18
      PRINT101,(D(I),I=1,IP)
       PRINT 17
      PRINT101,(DL(I),I=1,IP)
       PRINT 19
      PRINT101,(DU(I),I=1,IP)
      CALL PATTRN
       IK=0
      CALL FIND(TO)
      OBJO=OBJ(0)
      PRINT4,KL
4     FORMAT( '0NO. OF REDESIGN CYCLES=',I5)
      PRINT5,OBJO
5     FORMAT( '0OPTIMUM VALUE=',F13.8)
6     FORMAT( '0DESIGN PARAMETERS'/)
7     FORMAT(' P',I2,'=',F14.4)
      PRINT8
8     FORMAT( '0DESIGN VARIABLE VALUES'/)
```

```
      PRINT9,(K,D(K),K=1,IP)
9     FORMAT(' D',I1,'=',F15.8)
      IF(JP.EQ.0) GO TO 24
      PRINT10
10    FORMAT( 'ONEARNESS TO CONSTRAINTS'/)
      PRINT11,(K,G(K),K=1,JP)
 11    FORMAT(' G',I2,'=',F15.5)
 100   GO TO 24
33     STOP
      END
```

```
      SUBROUTINE PATTRN
        IMPLICIT REAL*8(A-H,O-Z)
      COMMON D(10),P(100),DZ(10),X(10),G(10),A,AMIN,DL(10),DU(10),
     1V(55,66),CK(10),Q,IP,JP,IA(10),KP,KW,KL,IK,IDUM(21),JX
       COMMON/OPT/BL(10),TN
       DIMENSION AD(10,10),EI(10,10),ET(10,10),AZ(10)
      TDIF=10.E+10
      IPP=IP
      AL=A
      IM=0
      STEST=0.
      KA=0
      KW=0
       KT=0
      KL=0
      KF=0
      KY=0
       KWW=0
       DO 112 K=1,JP
112    BL(K)=-0.1
      DO 111 K=1,IPP
      IF(D(K).LT.DL(K))D(K)=DL(K)
      IF(D(K).GT.DU(K))D(K)=DU(K)
  111 X(K)=D(K)
      CALL SIZE
25    KK=1
      DO 1 K=1,IPP
      DO 22 I=1,IPP
      EI(I,K)=0.
      ET(I,K)=0.
   22 AD(I,K)=0.
      ET(K,K)=1.
      EI(K,K)=1.
    1 CONTINUE
      ST=SB
      IF(KA.EQ.1)GO TO 9
       KA=1
190    IK=0
       JX=0
       CALL FIND(SB)
 9    ST=SB
       JX=0
      GO TO 100
 19    IK=0
       JX=0
       CALL FIND(ST)
 100  KL=KL+1
       IK=1
      KF=KF+1
      DO 2 J=1,IPP
      DO 17 K=1,IPP
      AZ(K)= EI(J,K)*A
      IF(KK.NE.0)AZ(K)=ET(J,K)*A
17    D(K)=D(K)+AZ(K)
      DO 35 K=1,IPP
      IF(D(K).GT.DU(K))GO TO 4
```

```
      IF(D(K).LT.DL(K))GO TO 4
   35 CONTINUE
      CALL FIND(SA)
      IF((ST-SA)/DABS(SB).GT.2.E-16) GO TO 5
    4 DO 18 K=1,IPP
18    D(K)=D(K)-AZ(K)-AZ(K)
      DO 36 K=1,IPP
      IF(D(K).LT.DL(K))GO TO 3
      IF(D(K).GT.DU(K))GO TO 3
   36 CONTINUE
      CALL FIND(SA)
      IF((ST-SA)/DABS(SB).LT.2.E-16) GO TO 3
    5 ST=SA
      GO TO 2
  3   DO 21 K=1,IPP
21    D(K)=D(K)+AZ(K)
  2   CONTINUE
      IF((SB-ST)/DABS(SB).LT.2.E-16) GO TO 7
26    SUM=0.
      DO 8 K=1,IPP
      DB=2.*D(K)-X(K)
      AD(1,K)=D(K)-X(K)
      IF(DABS(AD(1,K)).LT.+1.E-32) AD(1,K)=0
      SUM=SUM+AD(1,K)*AD(1,K)
      X(K)=D(K)
      SB=ST
      IF(DB.LT.DL(K))DB=DL(K)
      IF(DB.GT.DU(K))DB=DU(K)
    8 D(K)=DB
      SUM=DSQRT(SUM)
      IF(SUM.LE.0)GO TO 23
       IF(SUM-DABS(.5*A)) 7,7,950
  950 CONTINUE
      KK=0
      SUM=0.
      DO 11 J=1,IPP
      ET(1,J)=EI(1,J)
      EI(1,J)=AD(1,J)
   11 SUM=SUM+AD(1,J)**2
      SUM=DSQRT(SUM)
      DO 53 J=1,IPP
53    EI(1,J)=EI(1,J)/SUM
      DO 13 I=2,IPP
      K=I-1
      DO 14 J=I,IPP
   14 AD(I,J)=AD(1,J)
      SUMA=0.
      DO 15 J=1,IPP
      IF(DABS(EI(K,J)).LT.+1.E-32)  EI(K,J)=0
   15 SUMA=SUMA+EI(K,J)*AD(I,J)
      SUM=0.
      DO 16 J=1,IPP
      ET(I,J)=EI(I,J)
      EI(I,J)=AD(I,J)-EI(K,J)*SUMA
      IF(DABS(EI(I,J)).LT.+1.E-32) EI(I,J)=0
   16 SUM=SUM+EI(I,J)**2
```

```
      SUM=DSQRT(SUM)
      IF(SUM)27,27,28
27    DC 29 J=1,IPP
29    EI(I,J)=ET(I,J)
      GC TO 19
28    DC 13 J=1,IPP
   13 EI(I,J)=EI(I,J)/SUM
      GC TO 19
    7 DC 20 K=1,IPP
   20 D(K)=X(K)
      IF(KK)25,24,23
24    KK=-1
      IF(KF.GT.200)GO TO 23
       GO TO 9
23    CONTINUE
      TN=SB
      IF(JP.EQ.0)GO TO 32
      IK=0
       CALL FIND(SB)
      DO 30 I=1,JP
      IF(G(I).GT.-.01*BL(I))GO TO 31
30    CONTINUE
       IF(KWW.LE.3) KW=0
      KY=0
      GC TO 32
 31   KW=KW+1
      KY=KY+1
       KWW=KWW+1
       IK=0
       IF(KW.NE.0) CALL FIND(SB)
32     IF(KL.GT.KT+4) GO TO 33
       A=A/2.
       DO 34 K=1,JP
 34    BL(K)=BL(K)/2.
   33 CALL OPT2(SB,ST,JK,KY)
       KT=KL
      IF(JK.EQ.2) RETURN
 311  KF=0
       DO 300 I=1,JP
      IF(G(I).GT.-.01*BL(I))GO TO 301
300    CONTINUE
       IF(KWW.LE.3) KW=0
      KY=0
301   IF(JK.NE.0.) GO TO 46
      IF(A.EQ.AL) GO TO 26
       PRINT 114,A,SB,ST,(D(I),I=1,IP)
114    FORMAT( ' OPT2BASEA=',F11.9,'SB=',E16.8,'ST=',E16.8,'X=',10F10.6)
      AL=A
      DIF=DABS((SB-STEST)/SB)
      IF(DIF.GT..00001) GO TO 45
      IM=IM+1
      IF(IM.LT.2) GO TO 150
      IF(DIF.GT.TDIF) GO TO 45
      RETURN
 45    IM=0
 149  STEST=SB
```

```
      TDIF=DIF
      GO TO  26
148    IM=0
       A=.5*A
       DO 113 K=1,JP
 113   BL(K)=BL(K)/2.
46    DO 47 K=1,IPP
47    D(K)=X(K)
      IF(A.LT.AMIN) RETURN
      GO TO 190
 150  A=A/2
       DO 115 K=1,JP
 115   BL(K)=BL(K)/2.
      IF(A.LE.AMIN) RETURN
      GO TO 149
      END
```

```
      SUBROUTINE FIND(TO)
        IMPLICIT REAL*8(A-H,O-Z)
      COMMON D(10),P(100),DZ(10),X(10),G(10),A,AMIN,DL(10),DU(10),
     1V(55,66),CK(10),Q,IP,JP,IA(10),KP,KW,KL,IK,IL,IDUM(10),IC(10),JX
       COMMON/OPT/BL(10),TN
       DIMENSION DT(10)
      I=0
      TO=OBJ(0)
       TN=TO
      IF(JP.EQ.0) RETURN
      DO 25 K=1,IP
   25 DT(K)=D(K)
       IF(IK.EQ.0) GO TO 51
       IF(IL.EQ.0) RETURN
       DO 52 J=1,IL
       IB=IC(J)
       G(IB)=OBJ(IB)
       IF(G(IB)) 52,52,5
5      I=I+1
       IA(I)=IB
52     CONTINUE
       GO TO 56
51     IL=0
      DO 3 J=1,JP
      G(J)=OBJ(J)
      IF(G(J).LT.2.*BL(J))GO TO 3
6      IL=IL+1
       IC(IL)=J
      IF(G(J)) 3,3,4
    4 I=I+1
      IA(I)=J
    3 CONTINUE
56    IF(I)41,41,100
100    HTEMP=0.
      DO 21 K=1,I
       IB=IA(K)
       IF(KW.NE.0) GO TO  72
      IF(G(IB).LT.-BL(IB))GO TO 21
72    E=10000.*G(IB)
       IF(E.GT.HTEMP) HTEMP=E
21    CONTINUE
      IF(HTEMP.GT..001) GO TO 19
       IF(JX.NE.0) GO TO 42
      SUM=0.0
       JX=1
      DO 1 K=1,IP
      DT(K)=D(K)
      D(K)=D(K)+.001
      DZ(K)=(OBJ(0)-TN)/.001
      SUM=SUM+DZ(K)**2
1     D(K)=DT(K)
      SUM=DSQRT(SUM)
      DO 53K=1,IP
53    DZ(K)=.001*DZ(K)/SUM
42    DO 50 K=1,IP
   50 D(K)=DT(K)+ DZ(K)
```

```
      TA=OBJ(O)
      DO 18 K=1,L
      IB=IA(K)
      TT=OBJ(IB)
      IF (TT-G(IB)) 37,54,37
37    E=DABS((2.*(TO-TA)/(TT-G(IB)))*G(IB))
      IF(G(IB)+BL(IB))55,55,54
 54   E=10000.*G(IB)
55     IF(E.GT.HTEMP) HTEMP=E
  18 CONTINUE
19     CONTINUE
      TO=TO+HTEMP
      DO 26 K=1,IP
  26 D(K)=DT(K)
41    RETURN
      END
```

OPT2

```
      SUBROUTINE OPT2(SB,ST,JK,KY)
        IMPLICIT REAL*8(A-H,O-Z)
      INTEGER Z,E,G,BB,W,B,H
      COMMON D(10),F(100),DZ(10),X(10),T(10),V,AMIN,DL(1C),DU(10),
     1A(55,66),CK(10),Q,IP,JP,IA(10),KP,KW,KL,IK,IL,M,N,B,L,BB,E,G,Z,H,
     2W,IDUM(10),J5
       COMMON/OPT/BL(10),TP
      DIMENSION DT(10),SS(10),IB(10),IC(10),SSS(55),GG(1C),TN(11),DK(101
     1,20),ID(10),DBJ(11,10)
      VT=V/10.
       TQ=TP
      J6=0
      JIN=0
      JXN=0
       DEL=.00001
      IF(V.LT.DEL) DEL=V
       IPI=JP+4*IP+4
       DO 116 K=1,JP
       GG(K)=T(K)
116    ID(K)=0
117    DO 103 I=1,IP1
103    SSS(I)=0.
      JK=1
      N=2*IP+1.0
      N=IP+1
      Z=-1
      MM=JP+1
      M=MM+2*N+2*IP
      L=MM+N+IP
      E=0
       G=0
      NN=6
 90   Q=99999
 190  B=M+N+G+1
      W=M
 320  H=1
      BB=B+1
      J=0
      DC 4 I=1,IP
      IB(I)=0
      IC(I)=0
      IF(D(I)-DL(I).LT.V)GO TO 5
       IF(DU(I)-D(I).LT.V)GO TO 6
      GC TO 4
5     J=J+1
      IB(I)=J
       GO TO 4
6      J=J+1
      IC(I)=J
4      CONTINUE
      JX=J
      MM=JP+1
      IL=0
  156 J=0
      DC 22I=1,MM
      I1=I-1
```

```
      IF(I.EQ.1) TM=TQ
     IF(I.EQ.1) GO TO 77
      TM=GG(I1)
      IF(GG(I1).LE. BL(I1)) GO TO 22
      ID(I1)=1
   77 J=J+1
      TN(J)=TM
 22   CONTINUE
      IF(JX.EQ.JXN.AND.J.EQ.JIN) GO TO 14
      JIN=J
      JXN=JX
      DO 23 K=1,IP
      DT(K)=D(K)
      D(K)=D(K)+DEL
      J=0
      DO 24 I=1,MM
      I1=I-1
      IF(I.EQ.1) GO TO 777
      IF(GG(I1).LE.BL(I1)) GO TO 24
777   J=J+1
      IF(J6.NE.0)GO TO 778
      OBJ(I,K)=OBJ(I1)
 778  DK(J,K)=(OBJ(I,K)-TN(J))/DEL
 24   CONTINUE
 23   D(K)=DT(K)
      J6=1
       DO 25 I=1,J
      SUM=0.
      DO 26 K=1,IP
 26    SUM=SUM+DK(I,K)**2
       IF(SUM.EQ.0.) GO TO 25
       SUM=DSQRT(SUM)
       DO 27 K=1,IP
 27    DK(I,K)=DK(I,K)/SUM
 25    CONTINUE
       JQ=0
 330   LX=W+2
      IF(J.GT.1)GO TO 775
      IF(JX.EQ.0)GO TO 150
 775  DO 350 I=1,LX
 340  DO 350 J=1,B
 350  A(I,J)=0
      J=0
      DO 2 I=1,MM
      I1=I-1
      IF(I.EQ.1) GO TO 7
       IF(GG(I1).LE. BL(I1)) GO TO 2
7     J=J+1
       L=0
       DO 3 K=1,IP
      IF(IC(K).NE.0) GO TO 3
      L=L+1
      A(J,L)=DK(J,K)
3     CONTINUE
      DO 10 K=1,IP
      IF(IB(K).NE.0) GO TO 10
```

```
      L=L+1
      A(J,L)=-DK(J,K)
10     CONTINUE
      N=L+1
      IF(J.EQ.1)GO TO 2
      A(J,N)=CK(I1)
      IF(KY.NE.0)A(J,N)=10.
2      CONTINUE
      A(1,N)=1.
      JA=J+1
      DO 8 I=1,L
      J=J+1
8     A(J,I)=1.0
      W=J
      A(W+1,N)=1.0
      B=J+N+1
      BB=B+1
      DO 9 I=JA,W
 9    A(I,B)=1.0
      L=W
      DO 510 I=1,N
510   A(W+1,I)=FLOAT(Z)*A(W+1,I)
       M=J
      NN=6
      CALL LINPRO(K2)
       IF(K2.EQ.0) STOP
      LL=M+1
      DO 1460 I=1,LL
      J=A(I,BB)
       SSS(J)=A(I,B)
1460  CONTINUE
      L=0
      DO 19 I=1,IP
      IF(IC(I).NE.0)GO TO 12
      L=L+1
       SS(I)=SSS(L)
      GO TO 19
12     SS(I)=0
19     CONTINUE
      DO 13 I=1,IP
      IF(IB(I).NE.0)GO TO 13
      L=L+1
       SS(I)=SS(I)-SSS(L)
13    CONTINUE
   14 SUM=0.
      DO 11 I=1,IP
11    SUM=SUM+SS(I)*SS(I)
      SUM=DSQRT(SUM)
      IF(SUM.EQ.0.)GO TO 187
      DO 16 I=1,IP
      SS(I)=V*SS(I)/SUM
      D(I)=DT(I)+SS(I)
16    CONTINUE
18     IK=0
       CALL FIND(ST)
      IF(JP.EQ.0)GO TO157
```

```
      J=0
      DO 185 I=1,JP
      IF(T(I).LE.0.) GO TO 185
      IF(ID(I).EQ.0) BL(I)=-1.5*DABS(GG(I))
      IF(ID(I).EQ.0) J=1
185   CONTINUE
     IF(J.GT.0)GO TO 15
157  IF(SB.GT.ST) GO TO 101
187  V=.5*V
      DO 113 I=1,JP
113   BL(I)=BL(I)/2.
      IF(V.LT.VT) RETURN
     IF(V.LT.AMIN) RETURN
  15 DO 163 I=1,IP
163   D(I)=DT(I)
      GO TO 117
101    JK=0
      RETURN
150  SUMM=0.0
      DO 1 J1=1,IP
1    SUMM=SUMM+DK(I,J1)**2
      SUMM=DSQRT(SUMM)
     IF(SUMM.EQ.0) JK=2
     IF(SUMM.EQ.0.) RETURN
162  DO 151 I=1,IP
     D(I)=DT(I)-DK(I,I)*V/SUMM
     IF(D(I).LT.DL(I)) D(I)=DL(I)
151  IF(D(I).GT.DU(I)) D(I)=DU(I)
     SUMM=0.
     DO 152 I=1,IP
152   SUMM=SUMM+(D(I)-DT(I))**2
      SUMM=DSQRT(SUMM)
     DO 153 I=1,IP
153  D(I)=DT(I)+(D(I)-DT(I))*V/SUMM
      JK=0
     CALL FIND(ST)
     IF(SB.GT.ST)GO TO 101
     V=V/2.
      DO 112 I=1,JP
112   BL(I)=BL(I)/2.
      IF(V.LT.VT) RETURN
     IF(V.GT.AMIN)GO TO 162
     RETURN
     END
```

```
      SUBROUTINE SIZE
        IMPLICIT REAL*8(A-H,O-Z)
      COMMON D(10),P(100),DZ(10),X(10),G(10),A,AMIN,DL(10),DU(10),
     1V(55,66),CK(10),Q,IP,JP,IA(10),KP,KW,KL,IK,IDUM(22)
      TN=OBJ(0)
      DO 1 I=1,IP
      X(I)=D(I)
      D(I)=D(I)+.001
      DZ(I)=(OBJ(0)-TN)/.001
1     D(I)=X(I)
       DTEMP=0.0
       ITEMP=0
      DO 2 I=1,IP
      IF(DABS(DZ(I)).LT.DTEMP) GO TO 2
       DTEMP=DABS(DZ(I))
       ITEMP=I
2      CONTINUE
      A=DABS((0.01*TN)/DZ(ITEMP))
      IF(A.LT.0.005)A=0.005
      SUM=0.
      DO 3 I=1,IP
3      SUM=SUM+D(I)**2
      SUM=DSQRT(SUM)*.001
      IF(A.LT.SUM) A=SUM
      AMIN=A/100000.
      RETURN
      END
```

```
      SUBROUTINE LINPRO(K2)
        IMPLICIT REAL*8(A-H,O-Z)
      INTEGER Z,E,G,BB,W,B,R,C,H
      COMMON D(10),F(100),DZ(10),X(10),T(10),V,AMIN,DL(10),DU(10),
     1A(55,66),CK(10),Q,IP,JP,IA(10),KP,KW,KL,IK,IL,M,N,B,L,BB,E,G,Z,H,
     2W,IDUM(10),JX
      K2=1
      NA=6
      M=M-1.0
560   LL=M+2
      DO 580 K=2,LL
570   A(K-1,N+G+K-1)=1
580   A(K-1,BB)=K+N+G-1
600   IF(G.NE.0) GO TO 620
610   IF(E.EQ.0) GO TO 780
611   GO TO 650
620   KK=L+E+2
      LL=M+2
      DO 630 K=KK,LL
630   A(K-1,K+N-L-E-1)=-1
650   W=W+1
660   Q=0
670   LL=N+G
      DO 760 J=1,LL
680   S=0
690   LL1=M-G-E+2
      KK1=M+1
      DO 700 I=LL1,KK1
700   S=S+A(I,J)
720   A(W+1,J)=-S
730   IF(A(W+1,J).GT.Q) GO TO 760
740   Q=A(W+1,J)
750   C=J
760   CONTINUE
761   S=0
762   LL=M-G-E+2
      KK=M+1
      DO 763 J=LL,KK
763   S=S+A(J,B)
765   A(W+1,B)=-S
780     CONTINUE
790   IF(G.EQ.0) GO TO 810
      LL=N+1
      KK=N+G
810   IF(L.EQ.0) GO TO 830
      LL=N+G+1
      KK=N+G+L
      IC=1
830   IF(G+E.EQ.0) GO TO 2000
831   LL=N+G+L+1
      KK=B-1
      IC=1
860   GO TO 2000
895   IF(Q.EQ.99999) GO TO 1230
900   IF(Q.EQ.0) GO TO 1330
910   GO TO 1400
```

```
920   H=H+1
930   Q=.1E39
940   R=-1
      LL=M+1
950   DO 1000 I=1,LL
960   IF(A(I,C).LE.0) GO TO 1000
970   IF(A(I,B)/A(I,C).GT.Q) GO TO 1000
980   Q=A(I,B)/A(I,C)
990   R=I
1000  CONTINUE
1010  IF(FLOAT(R).GE.-.5) GO TO 1050
      IC=2
1030  GO TO 2000
1050  P=A(R,C)
1060  A(R,BB)=C
1070  DO 1080 J=1,B
1080  A(R,J)=A(R,J)/P
      LL=W+1
1100  DO 1180 I=1,LL
1110  IF(I.EQ.R) GO TO 1180
1120  DO 1170 J=1,B
1130  IF(J.EQ.C) GO TO 1170
1140  A(I,J)=A(I,J)-A(R,J)*A(I,C)
1150  IF(DABS(A(I,J)).GT..1E-4) GO TO 1170
1160  A(I,J)=0
1170  CONTINUE
1180  CONTINUE
      LL=W+1
1190  DO 1200 I=1,LL
1200  A(I,C)=0
1210  CONTINUE
1220  A(R,C)=1
1230  Q=0
1240  LL=N+G+L
      DO 1280 J=1,LL
1250  IF(A(W+1,J).GT.Q)GO TO 1280
1260  Q=A(W+1,J)
1270  C=J
1280  CONTINUE
1290  GO TO 900
1330  IF(W.EQ.M+1) GO TO 1360
1340  W=W-1
1350  IF(A(W+2,B).LT..1E-5) GO TO 1353
      K2=0
1352  RETURN
1353  LL=M+1
      DO 1358 I=1,LL
1354   IF(IDINT(A(I,BB)).LE.N+G+L) GO TO 1358
1355  DO 1356 J=1,B
1356  A(I,J)=0
1358  CONTINUE
1359  GO TO 1230
1360   CONTINUE
1400  IF(Q.EQ.0) GO TO 1420
1420  LL=M+1
1430  DO 1460 I=1,LL
```

```
1440   IF(IDINT(A(I,BB)).EQ.0) GO TO 1460
1460  CONTINUE
1470  IF(Q.NE.0) GO TO 920
1520  KK=N+1
      LL=B-G-1
      XJJ=-Z*A(W+1,B)
      LL=H-1
      IC=3
1550  GO TO 2000
2000  LL=H-1
      LL=W+1
      GO TO(895,1050,999),IQ
999   RETURN
      END
```

OBJ

```
      REAL FUNCTION OBJ*8(J)
       IMPLICIT REAL*8(A-H,O-Z)
      COMMON X(10),P(100),DUM(3693),IDUM(38)
101   OBJ=B-U
       IF(B.EQ.0. .OR.U.EQ.0.) RETURN
      OBJ=(B-U)/DABS(U)
       RETURN
      END
```

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
14 NJIT [redacted] NV-10

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
6 AN IMPROVED DIRECT SEARCH NUMERICAL OPTIMIZATION PROCEDURE.

5. TYPE OF REPORT & PERIOD COVERED
9 INTERIM rept.

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
10 MICHAEL PAPPAS

8. CONTRACT OR GRANT NUMBER(s)
15 ONR N00014-75-C-0987 new

9. PERFORMING ORGANIZATION NAME AND ADDRESS
NEW JERSEY INSTITUTE OF TECHNOLOGY
323 HIGH STREET
NEWARK, NEW JERSEY 07102

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
DR. NICHOLAS PERRONE, (CODE 474)
OFFICE OF NAVAL RES., 800 NO. QUINCY ST.
ARLINGTON, VIRGINIA 22217

12. REPORT DATE
11 FEBRUARY 1977

13. NUMBER OF PAGES
58 12 61p.

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
MATHEMATICAL PROGRAMMING; DESIGN, OPTIMAL
OPTIMIZATION, NUMERICAL, DESIGN, COMPUTER-AIDED.

392 024

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

AN IMPROVED, NONLINEAR, CONSTRAINED MATHEMATICAL PROGRAMMING OPTIMIZATION ALGORITHM COUPLING A ROTATING COORDINATE PATTERN SEARCH WITH A FEASIBLE DIRECTION FINDING PROCEDURE USED AT POINTS OF PATTERN SEARCH TERMINATION IS PRESENTED. THE PROCEDURE IS COMPARED WITH NINETEEN POPULAR ALGORITHMS, ON TEN TEST PROBLEMS IN WHICH THE MAJORITY OF CODES FAILED AT LEAST 50% OF THE TIME. ONLY THE NEW METHOD SOLVED ALL PROBLEMS, PARTICULARLY IN THE CASE OF CONSTRAINED PROBLEMS WHERE IT WAS BEST.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

6pg